DESIGN RATIONALE: FIT2099 SSB ASSIGNMENT 1

GROUP NAME JAVAPLUSPLUS

GROUP MEMBERS MOHAMED SHAKEEL MOHAMED RAFI 28021452

MATTI HADDAD 29708966

KERRY YUE SONG ZHENG 28794346

This document relays the design rationale and thought process for the tasks that were assigned.

1. Leave Affordance

Because none of the other classes can perform this type of affordance and each actor can perform the 'Leave action', having it as a subclass of HpAffordance will increase reusability, maintainability and reduce dependency as it will perform the action in its own module.

If an actor is holding an item (ring, wand, dagger, sword), the actor has the option of 'leaving' the item in the actor's current location. This action consumes one turn.

The Leave class is a subclass of HPAffordance that depends on:

- MessageRender to display a message to the user, a form of player feedback.
- HPEntetityInterface as it deals with entities and therefore the item needs to be managed by EntityManager. An HP item would not exist without the other.
- HPActor as actors are the ones who initiate the action.
- HPAction as to check if the action can be performed (an actor cannot give an item to an actor holding an item).

2. Give Affordance

Just as leave, each actor can perform the give action, having it as a subclass of HpAffordance will increase reusability, maintainability and reduce dependency as it will perform the action in its own module.

If an actor is holding an item (ring, wand, dagger, sword), the actor has the option of 'giving' the item to another actor if and only if both actors are in the same location and both are from the same team. In addition, the item can be rejected by the receiver, if so, it will stay with the actor that tried to 'give' it, this action consumes one turn.

The Give class is a subclass of HPAffordance that depends on:

- MessageRender to display a message to the user, a form of player feedback.
- HPEntetityInterface as it deals with entities and therefore the item needs to be managed by EntityManager.
- HPActor as actors are the ones who initiate the action.
- HPAction as to check if the action can be performed (an actor cannot give an item to an actor holding an item).

3. Wand and Spell Implementation

For a Spell to be Casted, there must be a Wand class and a Cast Action.

Furthermore, Cast may target both entities and actors, which will require further distinction.

3.1 Wand and Cast Class

A Wand, like a dagger or any other item in the game, can be picked up by any Actor. The existence of a Wand allows an Actor to Cast Spells onto other Actors, or itself.

3.2 Cast and Actor's Known Spells

Casting a spell requires a new action - Cast. This is different from Attack as it requires a Spell to be an input as well.

It will then execute the Spell's effect if the Actor knows the Spell.

Every actor will have a set of its known Spells.

3.3 Targeting Items

To allow Spells to be Casted on items, the interface HPEntityInterface is used in Cast and Spell to target both subclasses.

Additional checks must be implemented for individual spells to ensure that the target is the intended class type.

3.4 Expelliarmus

How expelliarmus works is that by casting this spell on a valid target i.e. a target that is not on the same team as the caster, by "forcing" the targeted actor to drop the item they are holding if they are holding one. The leave class would first check if the targeted actor is holding a item. If they are, it would remove the item the actor is carrying, by changing itemCarried attributes of hpactors, and place it in the location the targeted actor is at. As such, expelliarmus is dependent on leave. By using the leave class, it makes use of the existing affordance leave, and the functionality of expelliarmus is essentially an actor dropping an item (leave affordance) involuntarily.

3.5 Immobulus

The spell Immobulus when cast on an actor would render them unable to perform any actions such as moving, attacking or performing any affordances such as dropping or taking an item, essentially skipping one of their turns. In order to implement this function, we would ignore an actor's input to perform any actions for the duration of the spell.

4. Potions.

If an increaseHitPoints method is added to HPEntity, health potions could be implemented as class instances of HPEntity. But since potions are more interesting, e.g. magic, boost attack potions, it was

decided that it should be a subclass of HPEntity, it uses some of its functionality, in addition it will have a method increaseHealth.

Potions are used to allow an actor to replenish a random or a certain number of hit points if and only if the actor's health is not full and the actor could use them. Potions cannot increase the actor's health beyond its default health. Furthermore, potions are immediately consumed when an actor moves to the potion's location and do not consume the actor's turn.

5. Cast

Because its functionality behaves like the Attack class i.e. it uses same methods as the Attack class, it will therefore be a subclass of Attack, some methods will be overridden or added as long it doesn't break the contract with attack. Being a subclass of Attack, it will automatically inherit all dependencies of its parent class.

6. Dementor

Cannot be an instance of HPActor since it needs extra functionality, HPActor lacks this required functionality, therefore it will be made into a subclass of HPActor and it will override some methods to perform its functionality. It doesn't relate to Patrol class since its movement is random.

7. Spells enum class

An enum type is used to keep track of each actor's known spells, this is because the spell are predefined constants for each character. Furthermore, when cast is initiated it will need to refer to Spells enum class to perform its functionality.